گزارش فاز اول پروژه درس یادگیری ماشین

تهیه کننده: امیرحسین کارگران خوزانی

شماره دانشجویی: ۹۹۲۰۱۱۱۹

تیر ماه ۱۴۰۰

هدف پروژه و معرفی مساله و دادگان

- در این پروژه به دنبال استخراج بار معنایی و احساسی جملات هستیم. به گونه‌ای که مدل یادگیری ماشینی طراحی شود تا بتواند جملاتی که قبلا آن‌ها را ندیده است را به دو دسته مثبت و منفی تقسیم کند.
- دادگان در فایل dataset.csv به ما داده شده است. در این گام از پروژه در تمامی مراحل از ۸۰ درصد دادگان برای آموزش و صحت سنجی استفاده می‌شود و ۲۰ درصد برای تست گذاشته شده است.

تکنیک‌های اصلی استفاده شده

به طور کلی برای این که بتوان جملات را تجزیه و تحلیل کرد نیاز است که آن‌ها را به زبان ریاضی به مدل معرفی کنیم به همین خاطر از روش Bag of Words و Word to vec در ادامه استفاده شده است. در ادامه مختصرا به معرفی این دو روش می‌پردازیم.

- روش Bag of Words (BOW) : روشی است که در آن تنها کافی است که ابتدا کلمات یکتا در مجموعه کل جملات بدست آید. سپس برای هر جمله یک vector به اندازه کل کلمات ساخته شود و تعداد تکرار کلمه در آن جمله در جایگاه آن کلمه نوشته شود (منبع).
- به مثال زیر که از این لینک گرفته شده است توجه کنید:

ادامه تکنیک‌های اصلی استفاده شده: روش Bag of words

فرض کنید برای ۳ جمله زیر میخواهیم مدل BOW را محاسبه کنیم:

- **جمله اول:** من از این فیلم خوشم آمد
- **جمله دوم:** من این فیلم را به خاطر لیلی رشیدی دوست دارم
- **جمله سوم:** من از این فیلم خوشم نیامد.

سپس کلمات یکتا را بدست می‌اوریم و vector هایی برای هر ۳ جمله با تعداد تکرار هر کلمه می‌سازیم نتیجه به صورت زیر خواهد بود:

	من	از	این	فیلم	خوشم	آمد	را	به	خاطر	لیلی	رشیدی	دوست	دارم	نیامد
جمله اول	1	1	1	1	1	1	0	0	0	0	0	0	0	0
جمله دوم	1	0	1	1	0	0	1	1	1	1	1	1	1	0
جمله سوم	1	1	1	1	1	0	0	0	0	0	0	0	0	0

دقت شود اگر یکی از این کلمات در یک جمله چندبار تکرار می‌شد، همان عدد تکرار آن نوشته می‌شود.

ادامه تکنیک‌های اصلی استفاده شده: روش Word to Vec

- روش Word to Vec(W2V): روش قدرتمندی برای تبدیل هر کلمه به بردار می‌باشد که در سال ۲۰۱۳ توسط گوگل توسعه یافت. این روش خود از دو روش continuous bag-of-words (CBOW) و Skip-gram استفاده می‌کند. هر دو این روش‌های شبکه عصبی ساده هستند که بدون وجود لایه پنهانی به کمک چند قانون، بردارهای مورد نیاز را تولید می کنند.
  - در روش CBOW ابتدا به ازای هر لغت یک بردار با طول مشخص و با اعداد تصادفی تولید می شود. سپس به ازای هر کلمه از یک سند یا متن، تعدادی مشخص از کلمات بعد و قبل آنرا به شبکه عصبی می دهیم و با عملیات ساده ریاضی، بردار لغت فعلی را تولید می کنیم. با انجام این کار بر روی همه لغات بردارهای ما ساخته می‌شود.
  - در روش Skip-gram برعکس این روش یک لغت داده می‌شود و سعی می‌کند چند لغت قبل و بعد آنرا تشخیص دهد و با تغییر اعداد بردارهای لغات، نهایتا به یک وضعیت باثبات می رسد که بردارهای نهایی است (منبع).

بررسی روش‌های پیش پردازش دادگان

- در تمامی بقیه مراحل پروژه از تمامی دادگان استفاده شده است ولی در این قسمت ده هزار از دادگان استفاده شد. تمامی پیاده‌سازی های این مرحله در فایل 1_1.ipynb قرار دارد.

Part1

```
1 data_set = pd.read_csv('dataset.csv')
2 data_set = data_set[:DATASET_SIZE]
3 data_set.head(5)
```

	comment	sentiment
0	Oh my god, it just doesn't get any worse than ...	negative
1	If you're a layman interested in quantum theor...	negative
2	It's amazing that this no talent actor Chapa g...	negative
3	This must be one of the most overrated Spanish...	negative
4	Some critics have compared Chop Shop with the ...	positive

- ابتدا تمامی دادگان فرا خوانده شدند و نمایی از آن نمایش داده شد.

```
1 data_set.describe()
```

	comment	sentiment
count	10000	10000
unique	9983	2
top	Elfriede Jelinek, not quite a household name y...	negative
freq	2	5037

ادامه بررسی روش‌های پیش پردازش دادگان

- سپس برچسب negative به ۰ و برچسب positive به ۱ نگاشت شد.
- ۲۰ درصد دادگان برای تست و ۸۰ درصد برای آموزش و اعتبار سنجی انتخاب شدند.

```
X = data_set['comment']
Y = data_set['sentiment']


# label binarization
label_binarizer = LabelBinarizer()
Y = label_binarizer.fit_transform(Y)
Y = np.ravel(Y)
print(Y[:5])
```

```
[0 0 0 0 1]
```

Part 2,3

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=5)
```

ادامه بررسی روش‌های پیش پردازش دادگان

۳ روش برای بررسی پیش پردازش دادگان انتخاب شد که هر یک از این روش‌ها به کمک روش BOW و همراه با ۳ مدل یادگیری ماشین SVM، KNN و Logistic Regression ارزیابی شدند. از انجا که ممکن است تعداد کلمات یکتا خیلی زیاد باشند و در نتیجه ماتریس BOW که نظیر vector های همه جملات است بسیار بزرگ شود، پس یک محدودیت max_features قرار می‌دهیم که بردارها محدود به کلمات مهمتر شود و فرآیند یادگیری نیز سریعتر شود. به این منظور از تابع CountVectorizer در کتابخانه Sklearn استفاده می‌شود و بر روی دادگان آموزش این تابع fit می‌شود و بر روی دادگان تست تنها اعمال می‌شود.

دقت شود نباید این تابع بر روی کل دادگان زده شود و تنها باید بر روی مجموعه آموزش زده شود چرا که توزیع کلمات مهمتر در دادگان تست ممکن است بر روی نتایج تاثیر بگذارد و یک جور snooping رخ دهد!

در ادامه روش‌ها را با یکدیگر بررسی می‌کنیم:

ادامه بررسی روش‌های پیش پردازش دادگان: روش اول

- روش اول: در این روش تنها جملات tokenize می‌شوند ولی عملیات دیگری انجام نمی‌شود به این منظور یک پترن جامع برای قبول عددها و حروف و خاموش کردن lowercase به تابع CountVectorizer پاس داده می‌شود.

Part2: Without pre-processing

- lowercase is False
- Pattern: everything

```
count_vectorizer = CountVectorizer(lowercase = False, max_features=MAX_BOW_SIZE, token_pattern="[a-zA-Z0-9_'.]{1,}"
```

```
cv_X_train = count_vectorizer.fit_transform(X_train) #fit only over train data
cv_X_test = count_vectorizer.transform(X_test) #apply not fit!
```

ادامه بررسی روش‌های پیش پردازش دادگان: روش دوم

- روش دوم : در این روش علاوه بر tokenize شدن جملات کلمات lowercase می شوند و اعداد حذف و تنها کلمات با اندازه طول بزرگتر مساوی ۲ پذیرفته می‌شوند.

Part2: Elementry pre-processing

- lowercase is True
- Pattern: just words with lenght>1

```
count_vectorizer = CountVectorizer(lowercase=True, max_features=MAX_BOW_SIZE, token_pattern="[a-zA-Z_]{2,}")
```

```
cv_X_train = count_vectorizer.fit_transform(X_train) #fit only over train data
cv_X_test = count_vectorizer.transform(X_test) #apply not fit!
```

ادامه بررسی روش‌های پیش پردازش دادگان: روش سوم

- روش سوم: در این روش تابع Clean را توسعه دادیم که عملیات‌های کوچکتر کردن حروف، lemmatize کردن با توجه به حرف بودن یا فعل بودن، عملیات abbreviation برای چند تا از مهمترین اختصارها، حذف علائم نگارشی و حذف مجموعه stop word ها را انجام می‌دهد. همچنین مجموعه stop wordها هم مجموعه غنی شامل کتابخانه stop_words و کتابخانه nltk است.

Part2: Adcanced pre-processing ¶

- lowercase is True
- Pattern just words with lenght>1
- lemmatize
- stopwrods
- abbreviation

```
stop_words = list(get_stop_words('en'))
nltk_words = list(stopwords.words('english'))
stop_words.extend(nltk_words)

def lemmatize(text):
    list_pos = 0
    cleaned_str = ''
    lmtzr = WordNetLemmatizer()

    tagged_words = pos_tag(text)
    for word in tagged_words:
        if 'v' in word[1].lower():
            lemma = lmtzr.lemmatize(word[0], pos='v')
        else:
            lemma = lmtzr.lemmatize(word[0], pos='n')
        if list_pos == 0:
            cleaned_str = lemma
        else:
            cleaned_str = cleaned_str + ' ' + lemma
        list_pos += 1
    return cleaned_str

def clean(text):
    text = str(text).lower() #lowercase
    text = re.sub(r'\bid\b', 'i would', text) #start abbreviation
    text = re.sub(r'\bive\b', 'i have', text)
    text = re.sub(r'\bim\b', 'i am', text)
    text = re.sub(r'\bcant\b', 'can not', text)
    text = re.sub(r'\bdont\b', 'do not', text)
    text = re.sub(r'\bwont\b', 'will not', text)
    text = re.sub(r'\bthats\b', 'that is', text) #end abbreviation
    text = re.sub('[0-9]+', '', text) # delete numbers
    text = re.sub(r'[^\x00-\x7F]+',' ', text) #remove non-ascii
    text = re.sub('[<>()=~.,;:\\!?\\-()\\[\\]#/@"]+|[_x000D_]+|\u200c+|[\r\n]', ' ', text) #remove punctuations
    word_list = nltk.word_tokenize(text)
    text = lemmatize(word_list)
    word_list = text.split()
    word_list = list(filter(lambda word: word not in stop_words, word_list)) # delete stopwords
    word_list = [w for w in word_list if len(w)>1] # delete len = 1
    return word_list
```

```
count_vectorizer = CountVectorizer(tokenizer=lambda text: clean(text), max_features=MAX_BOW_SIZE)
```

```
cv_X_train = count_vectorizer.fit_transform(X_train) #fit only over train data
cv_X_test = count_vectorizer.transform(X_test) #apply not fit!
```

اجرای ۳ روش

- ۳ مدل یادگیری ماشین SVM، KNN و Logistic Regression به همراه چندین پارامتر در این مرحله انتخاب شدند.
- برای انجام عملات یادگیری از کتابخانه Gridsearch استفاده می‌شود این کتابخانه عملا همان مفهوم K-fold Cross Validation را انجام می‌دهد و سپس مدلی را از بین پارامترهای داده شده انتخاب می‌کند که بهترین scoring در میانگین همه fold ها را دارد. بهترین پارامتر هرکدام نیز در کد نیز گزارش شده است. از آنجا که تعداد داده کمتری را انتخاب کردیم من scoring را بر روی f1 گذاشتم که اگر توزیع بین جملات مثبت و منفی مناسب نیست scoring بایاسی همانند accuracy نداشته باشیم.

خروجی روش اول

```
-------SVM-------:
Best parameters set found on development set:
{'C': 10, 'kernel': 'rbf'}
Report Classification:
              precision    recall  f1-score   support

    positive       0.85      0.84      0.84      1010
    negative       0.83      0.85      0.84       990

    accuracy                           0.84      2000
   macro avg       0.84      0.84      0.84      2000
weighted avg       0.84      0.84      0.84      2000

Matrix Confusion:
 [[844 166]
 [151 839]]
Accuracy:
 0.8415
```

```
-------KNN-------:
Best parameters set found on development set:
{'n_neighbors': 21, 'weights': 'distance'}
KNeighborsClassifier(n_neighbors=21, weights='distance')
Report Classification:
              precision    recall  f1-score   support

    positive       0.67      0.49      0.57      1010
    negative       0.59      0.75      0.66       990

    accuracy                           0.62      2000
   macro avg       0.63      0.62      0.61      2000
weighted avg       0.63      0.62      0.61      2000

Matrix Confusion:
 [[498 512]
 [248 742]]
Accuracy:
 0.62
```

```
-------LR-------:
Best parameters set found on development set:
{'C': 1, 'penalty': 'l2'}
LogisticRegression(C=1)
Report Classification:
              precision    recall  f1-score   support

    positive       0.84      0.83      0.84      1010
    negative       0.83      0.84      0.84       990

    accuracy                           0.84      2000
   macro avg       0.84      0.84      0.84      2000
weighted avg       0.84      0.84      0.84      2000

Matrix Confusion:
 [[837 173]
 [155 835]]
Accuracy:
 0.836
```

خروجی روش دوم

```
-------SVM-------:
Best parameters set found on development set:
{'C': 10, 'kernel': 'rbf'}
Report Classification:
              precision    recall  f1-score   support

    positive       0.84      0.85      0.85      1010
    negative       0.85      0.84      0.84       990

    accuracy                           0.85      2000
   macro avg       0.85      0.85      0.85      2000
weighted avg       0.85      0.85      0.85      2000

Matrix Confusion:
 [[863 147]
 [160 830]]
Accuracy:
 0.8465
```

```
-------KNN-------:
Best parameters set found on development set:
{'n_neighbors': 28, 'weights': 'distance'}
KNeighborsClassifier(n_neighbors=28, weights='distance')
Report Classification:
              precision    recall  f1-score   support

    positive       0.71      0.49      0.58      1010
    negative       0.61      0.80      0.69       990

    accuracy                           0.65      2000
   macro avg       0.66      0.65      0.64      2000
weighted avg       0.66      0.65      0.64      2000

Matrix Confusion:
 [[499 511]
 [199 791]]
Accuracy:
 0.645
```

```
-------LR-------:
Best parameters set found on development set:
{'C': 1, 'penalty': 'l2'}
LogisticRegression(C=1)
Report Classification:
              precision    recall  f1-score   support

    positive       0.83      0.82      0.83      1010
    negative       0.82      0.83      0.83       990

    accuracy                           0.83      2000
   macro avg       0.83      0.83      0.83      2000
weighted avg       0.83      0.83      0.83      2000

Matrix Confusion:
 [[832 178]
 [169 821]]
Accuracy:
 0.8265
```

خروجی روش سوم

```
-------SVM-------:
Best parameters set found on development set:
{'C': 1, 'kernel': 'rbf'}
Report Classification:
              precision    recall  f1-score   support

    positive       0.86      0.83      0.84      1010
    negative       0.83      0.87      0.85       990

    accuracy                           0.85      2000
   macro avg       0.85      0.85      0.85      2000
weighted avg       0.85      0.85      0.85      2000

Matrix Confusion:
 [[835 175]
 [133 857]]
Accuracy:
 0.846
```

```
-------KNN-------:
Best parameters set found on development set:
{'n_neighbors': 28, 'weights': 'distance'}
KNeighborsClassifier(n_neighbors=28, weights='distance')
Report Classification:
              precision    recall  f1-score   support

    positive       0.68      0.61      0.64      1010
    negative       0.64      0.71      0.67       990

    accuracy                           0.66      2000
   macro avg       0.66      0.66      0.66      2000
weighted avg       0.66      0.66      0.66      2000

Matrix Confusion:
 [[613 397]
 [291 699]]
Accuracy:
 0.656
```

```
-------LR-------:
Best parameters set found on development set:
{'C': 1, 'penalty': 'l2'}
LogisticRegression(C=1)
Report Classification:
              precision    recall  f1-score   support

    positive       0.83      0.83      0.83      1010
    negative       0.83      0.83      0.83       990

    accuracy                           0.83      2000
   macro avg       0.83      0.83      0.83      2000
weighted avg       0.83      0.83      0.83      2000

Matrix Confusion:
 [[841 169]
 [167 823]]
Accuracy:
 0.832
```

نتیجه‌گیری اجرای ۳ روش

- بهترین روش یادگیری در میان این پارامترها SVM و سپس با اختلاف بسیار نزدیک Logistic Regression است و با اختلاف بیشتر KNN است.
- نتایج هر ۳ روش تقریبا به یکدیگر نزدیک است و روش سوم و دوم دقت بهتری دارند. در ادامه از روش سوم که در **مجموع** ۳ مدل بهتر از بقیه بهتر بوده است را انتخاب می‌کنیم.
- همچنین با افزایش دادگان اهمیت روشی مانند روش سوم بهتر مشخص می‌شود و همچنین این روش بدلیل بیشتر general بودنو وابسته نبودن به کلمات کم اهمیت احتمالا کمتر باعث overfit در مدل‌ها می‌شود.
- تمام این پیاده‌سازی ها تا به اینجا در فایل 1.ipynb وجود دارد.

بررسی تفاوت BOW و W2V

- بدین منظور از روش سوم پیش پردازش که در قسمت قبل توسعه دادیم استفاده می‌کنیم.
- تمامی پیاده سازی‌های این بخش در فایل 1_2.ipynb قرار دارد و بر روی همه ۴۵ هزار داده انجام شده است.
- پس از اعمال روش سوم بر روی دادگان و تقسیم آن‌ها به نسبت ۱ به ۵ برای دادگان تست و آموزش هر دو روش BOW و W2V را اعمال می‌کنیم و آن‌ها را با ۳ مدل می‌سنجیم.

روش BOW

BOW

```
count_vectorizer = CountVectorizer(tokenizer=lambda text: clean(text), max_features=MAX_BOW_SIZE)
```

```
cv_X_train = count_vectorizer.fit_transform(X_train) #fit only over train data
cv_X_test = count_vectorizer.transform(X_test) #apply not fit!
```

روش W2V

- تمام داده ها را به یکدیگر با \n می‌چسانیم و با همان همsplit می‌کنیم و سپس یک مدل Wordtovec از کتابخانه gensim می‌اوریم و بر روی جملات دادگان اموزش می‌سازیم.
- سپس برای هر جمله میانگین بردار کلمات ان‌ها را به عنوان نماینده آن جمله انتخاب می‌کنیم. اگر کلمه‌ای در مدل نیز وجود نداشت ان را skip می‌کنیم. این کار رو به طور مجزا برای دادگان اموزش و تست انجام می‌دهیم ولی مدل word2vec تنها بر روی دادگان آموزش ساخته شده است!

W2V

```
corpus_text_train = '\n'.join(X_train) # just for X_train fits
sentences_train = corpus_text_train.split('\n')
sentences_train = [clean(line) for line in sentences_train]

model = Word2Vec(sentences_train, window=5, min_count=3, workers=4)
vectors = model.wv

w2v_sentences_train = []
for index in range(len(sentences_train)):
    temp = []
    for word in sentences_train[index]:
        try:
            temp.append(vectors[word])
        except:
            pass
    w2v_sentences_train.append(np.mean(temp, axis=0))


corpus_text_test = '\n'.join(X_test) # just apply for X_test
sentences_test = corpus_text_test.split('\n')
sentences_test = [clean(line) for line in sentences_test]

w2v_sentences_test = []
for index in range(len(sentences_test)):
    temp = []
    for word in sentences_test[index]:
        try:
            temp.append(vectors[word])
        except:
            pass
    w2v_sentences_test.append(np.mean(temp, axis=0))
```

خروجی مدل KNN و پارامترها

knn: BOW ¶

```
k_range = list(range(1,31))
weight_options = ["uniform", "distance"]

param_grid = dict(n_neighbors = k_range, weights = weight_options)
knn = KNeighborsClassifier()

clf = GridSearchCV(knn, param_grid, scoring ='accuracy')
clf.fit(cv_X_train,Y_train)

print("-------KNN-------:")
print("Best parameters set found on development set:")
print (clf.best_params_)
print (clf.best_estimator_)
Y_test_pred = clf.predict(cv_X_test)
analysis(Y_test, Y_test_pred)
```

```
-------KNN-------:
Best parameters set found on development set:
{'n_neighbors': 28, 'weights': 'distance'}
KNeighborsClassifier(n_neighbors=28, weights='distance')
Report Classification:
              precision    recall  f1-score   support

    positive       0.66      0.69      0.68      4467
    negative       0.68      0.66      0.67      4533

    accuracy                           0.67      9000
   macro avg       0.67      0.67      0.67      9000
weighted avg       0.67      0.67      0.67      9000

Matrix Confusion:
 [[3075 1392]
 [1559 2974]]
Accuracy:
 0.6721111111111111
```

knn: W2V

```
k_range = list(range(1,31))
weight_options = ["uniform", "distance"]

param_grid = dict(n_neighbors = k_range, weights = weight_options)
knn = KNeighborsClassifier()

clf = GridSearchCV(knn, param_grid, scoring ='accuracy')
clf.fit(w2v_sentences_train,Y_train)

print("-------KNN-------:")
print("Best parameters set found on development set:")
print (clf.best_params_)
print (clf.best_estimator_)
Y_test_pred = clf.predict(w2v_sentences_test)
analysis(Y_test, Y_test_pred)
```

```
-------KNN-------:
Best parameters set found on development set:
{'n_neighbors': 24, 'weights': 'distance'}
KNeighborsClassifier(n_neighbors=24, weights='distance')
Report Classification:
              precision    recall  f1-score   support

    positive       0.80      0.84      0.82      4467
    negative       0.83      0.79      0.81      4533

    accuracy                           0.82      9000
   macro avg       0.82      0.82      0.82      9000
weighted avg       0.82      0.82      0.82      9000

Matrix Confusion:
 [[3748  719]
 [ 933 3600]]
Accuracy:
 0.8164444444444444
```

خروجی مدل Logistic Regression و پارامترها

logistic regression: Bow

```
grid_values = {'penalty': ['l2'], 'C': [0.1, 1,10, 100]}
clf = GridSearchCV(LogisticRegression(), param_grid=grid_values, scoring = 'accuracy')

clf.fit(cv_X_train,Y_train)

print("-------LR-------:")
print("Best parameters set found on development set:")
print (clf.best_params_)
print (clf.best_estimator_)
Y_test_pred = clf.predict(cv_X_test)
analysis(Y_test, Y_test_pred)
```

```
-------LR-------:
Best parameters set found on development set:
{'C': 0.1, 'penalty': 'l2'}
LogisticRegression(C=0.1)
Report Classification:
              precision    recall  f1-score   support

    positive       0.87      0.85      0.86      4467
    negative       0.86      0.88      0.87      4533

    accuracy                           0.86      9000
   macro avg       0.86      0.86      0.86      9000
weighted avg       0.86      0.86      0.86      9000

Matrix Confusion:
 [[3806  661]
 [ 560 3973]]
Accuracy:
 0.8643333333333333
```

logistic regression: W2V

```
grid_values = {'penalty': ['l2'], 'C': [0.1, 1,10, 100]}
clf = GridSearchCV(LogisticRegression(), param_grid=grid_values, scoring = 'accuracy')

clf.fit(w2v_sentences_train,Y_train)

print("-------LR-------:")
print("Best parameters set found on development set:")
print (clf.best_params_)
print (clf.best_estimator_)
Y_test_pred = clf.predict(w2v_sentences_test)
analysis(Y_test, Y_test_pred)
```

```
-------LR-------:
Best parameters set found on development set:
{'C': 1, 'penalty': 'l2'}
LogisticRegression(C=1)
Report Classification:
              precision    recall  f1-score   support

    positive       0.86      0.85      0.85      4467
    negative       0.85      0.86      0.86      4533

    accuracy                           0.86      9000
   macro avg       0.86      0.86      0.86      9000
weighted avg       0.86      0.86      0.86      9000

Matrix Confusion:
 [[3787  680]
 [ 620 3913]]
Accuracy:
 0.8555555555555555
```

خروجی مدل SVM و پارامترها

svm: BOW

```
tuned_parameters = [{'kernel': ['poly', 'rbf'], 'C': [1, 10]}]

clf = GridSearchCV(SVC(), tuned_parameters, scoring ='accuracy')
clf.fit(cv_X_train, Y_train)

print("-------SVM-------:")
print("Best parameters set found on development set:")
print(clf.best_params_)
Y_test_pred = clf.predict(cv_X_test)
analysis(Y_test, Y_test_pred)

```

```
-------SVM-------:
Best parameters set found on development set:
{'C': 1, 'kernel': 'rbf'}
Report Classification:
              precision    recall  f1-score   support

    positive       0.87      0.84      0.86      4467
    negative       0.85      0.88      0.86      4533

    accuracy                           0.86      9000
   macro avg       0.86      0.86      0.86      9000
weighted avg       0.86      0.86      0.86      9000

Matrix Confusion:
 [[3772  695]
 [ 557 3976]]
Accuracy:
 0.8608888888888889
```

svm: W2V

```
tuned_parameters = [{'kernel': ['poly','rbf'], 'C': [1, 10]}]

clf = GridSearchCV(SVC(), tuned_parameters, scoring ='accuracy')
clf.fit(w2v_sentences_train, Y_train)

print("-------SVM-------:")
print("Best parameters set found on development set:")
print(clf.best_params_)
Y_test_pred = clf.predict(w2v_sentences_test)
analysis(Y_test, Y_test_pred)

```

```
-------SVM-------:
Best parameters set found on development set:
{'C': 10, 'kernel': 'rbf'}
Report Classification:
              precision    recall  f1-score   support

    positive       0.87      0.85      0.86      4467
    negative       0.86      0.87      0.86      4533

    accuracy                           0.86      9000
   macro avg       0.86      0.86      0.86      9000
weighted avg       0.86      0.86      0.86      9000

Matrix Confusion:
 [[3818  649]
 [ 589 3944]]
Accuracy:
 0.8624444444444445
```

نتیجه‌گیری اجرای BOW و W2V

- در تمامی مدل‌ها دقت W2V بهتر از دقت BOW است.
- در KNN با اختلاف زیاد W2V بهتر از BOW است.
- در Logistic Regression با روش W2V بهترین دقت بین همه مدل‌ها را کسب کرده است.

بخش امتیازی

- در این بخش از GoogleNews-vectors-negative300 که یک مدل pre-trained شده است استفاده کردیم. قاعدتا چنین مدلی چون بر روی متن‌هایی بیشتری یادگرفته شده است باید بردارهای مناسب‌تری و با مفهوم‌تری برای کلمات ارائه دهد.

تمام پیاده‌سازی این بخش در 1_3.ipynb پیاده سازی شده است.

خروجی اجرا با چندین پارامتر مختلف و انجام عملیات cross-validation و تست بر روی مجموعه دیده نشده

MLP: W2V

```
mlp_gs = MLPClassifier(max_iter=100)
parameter_space = {
    'hidden_layer_sizes': [(50,50,50), (50,100,50),(10,30,10),(20,), (50,), (100,), (150,)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05, 0.1],
    'learning_rate': ['constant','adaptive'],
}
clf = GridSearchCV(mlp_gs, parameter_space, n_jobs=-1, cv=5, scoring='accuracy')
clf.fit(w2v_sentences_train, Y_train)

print("-------MLP-------:")
print("Best parameters set found on development set:")
print (clf.best_params_)
print (clf.best_estimator_)
Y_test_pred = clf.predict(w2v_sentences_test)
analysis(Y_test, Y_test_pred)
```

```
-------MLP-------:
Best parameters set found on development set:
{'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': (50,), 'learning_rate': 'constant', 'solver': 'adam'}
MLPClassifier(hidden_layer_sizes=(50,), max_iter=100)
Report Classification:
              precision    recall  f1-score   support

    positive       0.84      0.87      0.85      4467
    negative       0.87      0.84      0.85      4533

    accuracy                           0.85      9000
   macro avg       0.85      0.85      0.85      9000
weighted avg       0.85      0.85      0.85      9000

Matrix Confusion:
 [[3883  584]
 [ 738 3795]]
Accuracy:
 0.8531111111111112
```

نتیجه‌گیری

- به اندازه روش‌های قبلی دقت خوبی را روش MLP بر روی دادگان تست کسب کرد.